Machine Learning Final Project

Team: Anything

李文鼎
電機四 B98901093

向思蓉
電機所 R01921050

郭佳翰
電機四 B98901020

***Abstract*—In the machine learning competition, we were overwhelmed by the large dataset and some missing values. To, cope with this, we used Expectation-Maximization algorithm and Interpolation to deal with missing data. For the sake of time efficiency, we preprocessed the training data with feature selection and under sampling. Furthermore, we tried Naïve Bayes, Linear Regression, Logistic Regression, Support Vector Machines using RBF Kernel, Random Forest Regression and so on. Finally, we blended the predictions from those algorithms and get the best result (AUC on scoreboard = 0.771108).**

***Keywords*—rank, classification, data process**

I. INTRODUCTION

In this competition, we were asked to predict whether customers would click an ad provided by a search engine. Training and test data were provided by the class. There is a scoreboard that we can upload our predictions and get the AUC evaluations (area under the ROC curve) of our predictions on half of the test data. We decided to use cross validations to get some parameters and chose the one with highest score on scoreboard as the best choice.

II. MISSING DATA IMPUTATION

In the original dataset, we can easily find that there are many missing values (about 10%). Because the missing data account for a great proportion and every dimension has missing values, it is impossible that we just throw it away. By the way, if we just make imputation by replacing missing values with mean, it will get really bad performance. Therefore we have implemented the following methods.

A. Expectation and Maximization

A general approach for computing maximum likelihood estimates from incomplete data is so-called EM algorithm [7], which consists of iterative calculation involving two steps.

1. Prediction Step.
   Given some estimate $\tilde{\theta}$ of the unknown parameters, predict the contribution of any missing observation to the sufficient statistics.

$$\tilde{\mathbf{x}}_j^{(1)} = \mathrm{E}\left(\mathbf{x}_j^{(1)} \middle| \mathbf{x}_j^{(2)}; \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}\right) = \tilde{\boldsymbol{\mu}}^{(1)} + \widetilde{\boldsymbol{\Sigma}_{12}}\widetilde{\boldsymbol{\Sigma}_{22}}^{-1}\left(\mathbf{x}_j^{(2)} - \tilde{\boldsymbol{\mu}}^{(2)}\right)$$

$$\widetilde{\mathbf{x}_j^{(1)}\mathbf{x}_j^{(1)'}} = \mathrm{E}\left(\mathbf{x}_j^{(1)}\mathbf{x}_j^{(1)'} \middle| \mathbf{x}_j^{(2)}; \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}\right) = \widetilde{\boldsymbol{\Sigma}_{11}} - \widetilde{\boldsymbol{\Sigma}_{12}}\widetilde{\boldsymbol{\Sigma}_{22}}^{-1}\widetilde{\boldsymbol{\Sigma}_{21}} + \tilde{\mathbf{x}}_j^{(1)}\tilde{\mathbf{x}}_j^{(1)'}$$

$$\widetilde{\mathbf{x}_j^{(1)}\mathbf{x}_j^{(2)'}} = \mathrm{E}\left(\mathbf{x}_j^{(1)}\mathbf{x}_j^{(2)'} \middle| \mathbf{x}_j^{(2)}; \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}\right) = \tilde{\mathbf{x}}_j^{(1)}\mathbf{x}_j^{(2)'}$$

2. Estimation Step.
   Use the predicted sufficient statistics to compute the revised estimate of the parameters.

$$\tilde{\boldsymbol{\mu}} = \overline{\mathbf{X}} \text{ and } \tilde{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_{j=1}^{n}\mathbf{X}_j\mathbf{X}_j' - \tilde{\boldsymbol{\mu}}\tilde{\boldsymbol{\mu}}'$$

B. Interpolation and Extrapolation

In this method, we use extra/interpolation [8] to replace the missing values. Missing data interpolation is a particular case of data regularization, where the input data are already given on a regular grid, and one needs to reconstruct only the missing values in empty bins.

From table 1, we can find that data imputation using EM algorithm has better performance than Interpolation. It is because that interpolation only uses linear model to replace the missing values. As a result, we use data replaced with EM algorithm in the following experiments. By the way, scaled data gets worse performance than normal data. There may be a reason that we scale up the noisy data and therefore get a worse performance.

TABLE I. MISSING DATA WITH LOGISTIC REGRESSION

Method	AUC evaluation	
	Training AUC	*Score Board AUC*
EM	0.7476	0.7633
EM*	0.7477	0.7632
Interpolation	0.7462	0.7579
Interpolation*	0.7460	0.7576

III. FEATURE SELECTION

For reason of data interpretation and reduction computation time, we had no choice but to make feature selection and the result is surprisingly good. The reason may be that the noisy attributes are filtered out and we only use the most informative attributes. The following are some methods we implement.

A. Principle Component Analysis

We use principle component analysis [9] to implement orthogonal data transformation, which makes the variance of the projected data to be maximum value. In PCA, The variance of the projected data (transformed by projection vectorω) which we want to maximize is

$$\mathrm{Var}(\omega^{\mathrm{T}}\mathrm{x}) = \mathrm{E}(\omega^{\mathrm{T}}\mathrm{x} - \omega^{\mathrm{T}}\mu)^2 = \omega^{\mathrm{T}}\sum\omega$$

By some computation, the projected axes (eigenvectors) ω are the solution of $\Sigma\omega = \lambda\omega$. ($\Sigma$ is the covariance matrix of the original data) Furthermore, we selected those eigenvectors (In

our experiment, 10) with eigen value larger than 1, and the corresponding reconstruction rate

$$\text{construction rate} = \frac{\text{sum(selected eigen value)}}{\text{sum(total eigen value)}}$$

is about 0.99. We used the projected data as one of our dataset.

B. Predictive Ability and Redundancy

From result of some data analysis, we can detect that there are some dependencies in the original 71-dim data which may lead to high redundancy and bad training result. As a result, we preferred dataset with high correlation with class (high predictive ability) label but low correlation between each two dimensions (low redundancy) as our ideal processed training source. With the help of CfsSubsetEval filter of WEKA [10], we use full data and 5 cross validation set respectively to get the appropriate subset of the original dataset.

C. Information Gain

Information gain, by definition, is the change in information entropy from a prior state to a state that takes some information as given.

$$\text{IG}(\text{class}, \text{attribute}) = \text{H}(\text{class}) - \text{H}(\text{class}|\text{attributes})$$

With the help of InfoGainAttributeEval filter of WEKA [10], we use full data and 5 cross validation set respectively to get the appropriate subset of the original dataset.

We use the above methods to implement feature selection, and Table 2 shows that the performance using EM, logistic regression and one of the feature selection method together. We can find that there is a trend that more attributes will result in better performance, except PCA. Surprisingly, the performance of PCA with 10-dim transformed data is better than the one with 20-dim and even 71-dim data. It is probably because that PCA makes the eigenvector with larger eigen value more important, thus the other eigenvectors may be noisier. Besides, the performance of 25-dim data selected by CfsSubsetEval is almost the same as that of 39-dim data selected by InfoGainAttributeEval. There may be a reason that CfsSubsetEval stepwise optimizes the criterion, but InfoGainAttributeEval uses greedy algorithm.

IV. SAMPLING

A. Under-Sampling

It is obvious that the number of +1 and -1 label in the training set is extremely unbalanced (1816/38184). For a binary classification, if the class labels are heavily unbalanced, then the prediction will be heavily biased to majority class. To avoid this biased problem, we sample the dataset with equal +1 and -1 labels. That is to say, we filter out some data of the majority class to make a sub training set.

B. Over-Sampling

There is another way to make a balanced data set from an unbalanced data set. If we duplicate a minority data many times to generate a bigger data set, we can make the number of data of each class roughly the same. We mostly use under-sampling method since using small data set during training is more efficient. According to [11], there is no major difference between using over-sampling, under-sampling or using both at the same time. Therefore, we choose under-sampling, which is the most efficient. We could repeat the under-sampling and training procedure multiple times to obtain different models, and then choose the best one based on the results of cross validation or doing blending. Therefore, by using under-sampling which results in small training set, we would not lose much information.

TABLE II. FEATURE SELECTION WITH LOGISTIC REGRESSION AND EM

Method	AUC evaluation	
	Training AUC	***Score Board AUC***
PCA$^{(10)}$	0.7405	0.7629
PCA$^{(20)}$	0.7427	0.7604
Predictive Ability$^{(\text{all } 16)}$	0.7437	0.7606
Predictive Ability$^{(5\text{CV } 25)}$	0.7454	0.7631
Information Gain$^{(\text{all } 25)}$	0.7439	0.7635
Information Gain$^{(\text{all } 39)}$	0.7454	0.7634
Information Gain$^{(5\text{CV } 41)}$	0.7455	0.7631

V. VALIDATION

We use 5-fold cross validation and take the result on scoreboard as our validation method. We find that the result of 5-fold cross validation has a high variance, that is, the 5 validation AUC from the 5-fold cross validation could be very different. The difference between maximum and minimum of AUC can be as high as 0.05.

We think it is because the number of data with label +1, 1816, is already small, and it gets even smaller if further divided by 5. Although the number of label -1 data is big, small number of label +1 data would make the result not close to the final test set AUC. Since the number of label +1 data is small, we think we cannot afford to let the validation set bigger, which would result in smaller number of +1 label in training set. Thus, we believe that the cross validation does not accurately reflect the test-AUC. We think that if we want to improve our performance on the scale of 0.001~0.01, then cross validation is not really helpful. Therefore, we use cross validation to ensure that the performance pass our base line (AUC=0.73~0.74). We would roughly find the parameter which gives us good results on cross validation and then we fine tune the parameter by using the scoreboard result.

However, using the result on the scoreboard may be accompanied by the risk of over-fitting. We think the public test set on the scoreboard is quite big as it is roughly on the same scale as number of training data, and equals to the number of hidden test data. As a result, we believe we can trust the scoreboard result and use it as validation to improve our performance as long as we do not use slow cheating to get the public test set data.

To address these problems, we plot the figure as follows: We assume that we use scoreboard to do model selection and select the model which performs best on the public test set, and we plot the hidden test set AUC of that model accordingly. These results are obtained from the scoreboard. We can see that the public test set AUC is higher than the hidden test set AUC. However, as we improve our public test set AUC, the hidden test set AUC improves as well most of the time. Thus, we can say that the method we use is not over-fitting badly. Overall, using cross validation and then using scoreboard as validation indeed improve our hidden test AUC.

VI. CLASSIFICATION

A. Naïve Bayes

Naïve Bayes [12] assumes that the behavior of a particular feature of a class is unrelated to that of the other feature of the class, when given the class. Naïve Bayes has an advantage that it only requires a small amount of time to get the parameters, which are means and variances of each feature. Calculating parameters in aspect of statistical, naïve Bayes can ignore missing values in the dataset, which means that we can avoid the uncertainty caused by replacing missing values. Assuming that every pairs of features are independent, we can get

$$P(y = c|\boldsymbol{x}) = \frac{1}{Z} P(y = c) \prod_{i=1}^{d} P(x_i|y = c)$$

Implementing naïve Bayes in our situation, we can get the expectation:

$$\begin{aligned} E(\boldsymbol{x}) &= -1 * P(y = -1|\boldsymbol{x}) + 1 * P(y = 1|\boldsymbol{x}) \\ &= 2 * \frac{1}{Z} P(y = 1) \prod_{i=1}^{d} P(x_i|y = 1) - 1 \end{aligned}$$

Noted that $\frac{2}{Z}$ and -1 can be ignored because AUC only takes the order of the samples into consideration. In the training step, we calculate $P(y = 1) = (\#\ of\ samples\ in\ class\ 1)/(total\ \#\ of\ samples)$, μ_i and σ_i, which are means and variances of each feature of the samples in class 1. Missing values were skipped when calculating means and variances. In the step of test, for each sample and each feature, we evaluate

$$P(x_i|y = 1) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}}$$

Then we calculate the prediction score of a sample

$$P(y = 1) \prod_{\{i:\, x_i\ is\ not\ missing\}} P(x_i|y = 1)$$

The probability of the feature with missing value is ignored [4]. The model achieves 0.7467 AUC on the public scoreboard.

B. Linear Regression

Linear regression uses the hypothesis set that contains all kinds of linear combinations of the components of **x** and tries to minimize the squared error $E_{in}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(\boldsymbol{x}_n) - y_0)^2$. For tuning, we decided to use ridge regression. The solution can be calculated simply by the formula: $w_{reg} = (Z^T Z + \lambda I)^{-1} Z^T \boldsymbol{y}$. It has perfect computation efficiency and there is no parameter other than λ for regularization, which means we can get the best choice of parameter easily.

Trying λ from 0 to 60 with 0.5 step size, Figure 1 shows the best choice is $\lambda = 22.5$, and the performance of ridge regression on scoreboard is AUC=0.7615.

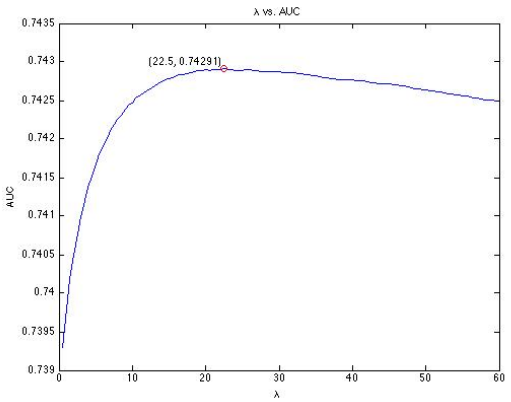

Fig. 1. Ridge Regression: λ versus AUC

C. Logistic Regression

Given input data **x** and weight **w**, the logistic regression algorithm models the classification model by

$$\mathrm{P}(\mathrm{y} = \pm 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1+\exp\,(-\mathrm{y}(\mathbf{w}^{\mathrm{T}}\mathbf{x}))},$$

where y is the classification label. To reduce model complexity and to avoid over-fitting, we add the regularization term $\frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$ into the minimization criterion

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + \mathrm{C}\sum_{\mathrm{i}=1}^{\mathrm{N}} \log\,(1 + \exp\,(-\mathrm{y_i}(\mathbf{w}^{\mathrm{T}}\mathrm{x_i}))),$$

where C is the regularization parameter. We use the scikit-learn package [5] to solve the problem. The model achieves 0.7616 AUC on public scoreboard and 0.7402 AUC on the hidden scoreboard.

D. Random Optimization with Linear Model

If we use linear model, it is better to optimize in-sample AUC rather than other error functions. Because AUC is not differentiable and not convex, we use a naive method commonly used when we have no idea about the function we want to optimize. We randomly initial w, and then obtain v from Gaussian distribution (0,0.001). If w'=w+w.*v is better than w in terms of AUC, update w by w'. [13]

E. K Nearest Neighbor

K-nearest neighbor is the method to find the most similar k training data of a test data point and to weight the k data linearly as output result. We use scikit-learn package [5], which has k-nearest-neighbor implementation. The similarity function has something to do with Euclidean distance of two data on the feature space. There are two options for weight,

uniform and distance. The uniform weighted option uniformly sums up the k neighbors, and the distance-weighted option uses the inverse of distance between two data on the feature space as weight. For the classification, we output the percentage of k neighbors being +1; for the regression, we output the linearly weighted sum of label y as output. We use EM algorithm to recover missing data and then use scikit-learn [5] to perform k-nearest neighbor algorithm. The model achieves 0.7645 AUC on the public scoreboard.

TABLE III. K NEAREST NEIGHBOR OF DIFFERENT K AND CORRESPONDING SCOREBOARD AUC

Without under-sampling		With under-sampling			
weighted 1/distance		Weighted		uniform wieghted	
K	Scoreboard AUC	K	Scorebo-ard AUC	K	Scorebo-ard AUC
e+04	0.7632	1000	0.7632	300	0.7633
4000	0.7630	1500	0.7628	270	0.7636
2000	0.7628	1204	0.7629	250	0.7645
1000	0.7592	900	0.7632	--	--
700	0.7568	300	0.7624	--	--

F. Support Vector Machine

We chose C-SVM and RBF (radial basis function) kernel. We used RBF kernel because it only needs one parameter for us to find the best one and its numerical range is always in [0, 1], which means it is more numerically stable than polynomial and linear kernel [1][2].

We used libsvm [3] as our analyzing tool. In order to make use of "grid.py" tool in the package, we change some code and make AUC instead of precision as the output. First, we tried parameter c in logistic scale from 2^0 to 2^{40}, γ in logistic scale from 2^{-30} to 2^{30} and the step size in log scale is 5. Figure 2 shows that the best choice seem to locate in the region that $c \in [2^{10}, 2^{25}], \gamma \in [2^{-30}, 2^{-10}]$. Therefore, we tried parameter c in log scale from 2^{10} to 2^{25}, γ in logistic scale from 2^{-30} to 2^{10} and the step size in logistic scale is 1. According to Figure 3, the best choices of parameters are $c = 2^{19}$ and $\gamma = 2^{-26}$ and we can achieve 0.7614 AUC on the scoreboard.

G. Random Forest

Random forest is an ensemble method constructed from decision tree. We use random forest in the scikit-learn package, which uses CART (classification and regression tree) for decision tree [15]. The CART algorithm splits the source set into subsets based on minimizing impurity function (We use Gini function in the as impurity function.) This process is repeated on each subset in a recursive manner called recursive partitioning. Then, we use boosting method, which uses sampling with replacement to construct different training set and to aggregate decision trees.

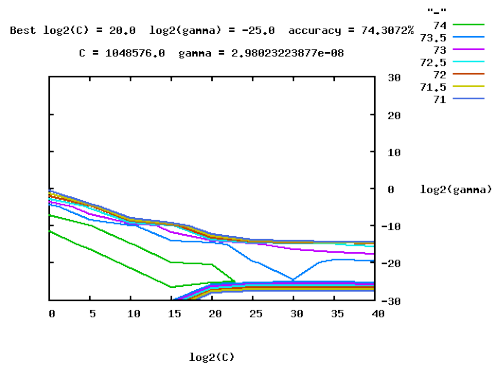

Fig.2. result of libsvm grid

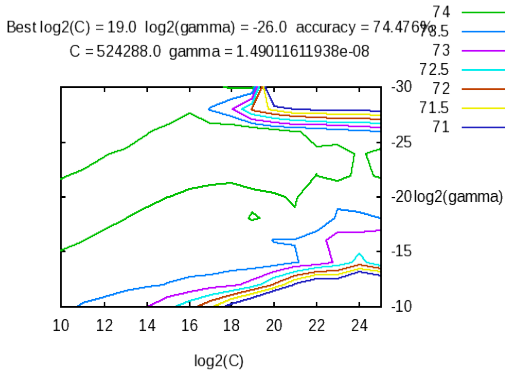

Fig.3. result of libsvm grid (preciser)

By the way, when reaching terminals (leaves), the probability measure we use for classification tree of each class is proportional to the number of each class in the leaf. To get the better AUC on scoreboard, we use probability measure instead of +1/-1label as our output. For regression tree, we use minimum square error improvement as impurity function[].

TABLE IV. RANDOM FOREST WITH 200 DECISION TREES

	Classification Criterion		***Regression Criterion***	
	With US	***Without US***	***With US***	***Without US***
depth	***CV AUC***	***CV AUC***	***CV AUC***	***CV AUC***
1	0.7298	0.7066	0.7303	0.6896
2	0.7363	0.7418	0.7393	0.7454
3	0.7377	0.7435	0.7358	0.7466
4	0.7375	0.7438	0.7348	0.7455
5	0.7373	0.7444	0.7335	0.7447
6	0.7376	0.7438	0.7339	0.7447
7	0.7360	0.7434	0.7320	0.7440
8	0.7344	0.7421	0.7322	0.7428

We used cross validation to determine the range of parameter we want to focus. As in Table 4, we can see that the number of tree is not really matter much, so we set the number of tree to 200 and the maximum tree depth should be 2~4, not too large. With these rules in mind, we can then tune the

parameters (e.g. minimum number to split a node for regularization) using public test set score on the scoreboard.

In the preprocessing step, we used the EM algorithm to recover missing values, under-sampled different dataset, and chose some datasets that perform well on cross validation set and scoreboard. Training on the under-sampling data set with equal number of y=+1 and y=-1 data, Random Forest Regression (n_estimators=200, max_depth=4, min_samples_split=750, min_samples_leaf=437, n_jobs=7) achieves 0.7701 on the public scoreboard and 0.7521 on the hidden scoreboard. In addition, Random Forest Classifier (n_estimators=2000, max_depth=6) achieves0.7669 on the public scoreboard and 0.7478 on the hidden scoreboard.

We can use random forest to do feature selection as well. For each node we can add importance on a feature as follows. If using Gini function as impurity function, the difference of the original error and the error of the split times the number of samples that passed the node will be added to that feature. If using squared function, improvement in squared error will be added [16].

Besides, by computing feature importance, we can see that the 25th feature is the most important feature in using random forest. Therefore we think if we heavily depend on that feature, missing values on that feature would cause high error. To dealing with this problem, we thought we can construct two models. We construct two datasets, one is those data whose 25th feature is not missing and the other one is using all data but filtering out the 25th feature such that there are only 70 features left. By doing so, we train two random forest model h1, h2 accordingly and use h(x) below as the prediction function.

$$\mathrm{h(x)} = \begin{cases} \mathrm{h_1(x)}, \text{if the 25th feature not missing} \\ \mathrm{h_2(x)}, \text{if the 25th feature missing} \end{cases}$$

With that, we can avoid relying on missing 25th feature, which is actually help to improve the performance of random forest. The model achieves 0.7683 on the public score board and 0.7487 on the hidden score board.

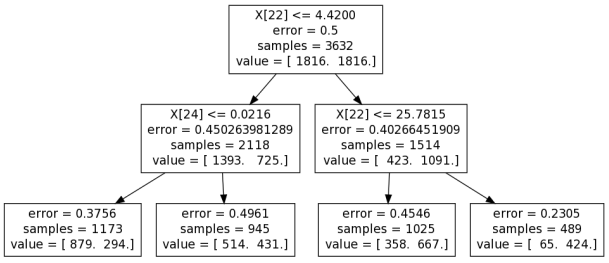

Fig.4. One of the decision tree generated by random forest.

H. Matrix Factorization

Feature-based matrix factorization [18] is an abstract matrix factorization model that uses features to describe the global bias and user/item factors.

$$\widehat{y_{\mu,\mathrm{I}}} = \mu + \left(\sum_{\mathrm{g}\in\mathrm{G}} \gamma_\mathrm{g} b_\mathrm{g} + \sum_{\mathrm{m}\in\mathrm{M}} \alpha_\mathrm{m} b^\mathrm{i}{}_\mathrm{m} + \sum_{\mathrm{n}\in\mathrm{N}} \beta_\mathrm{n} b^\mathrm{i}{}_\mathrm{n}\right) + \left(\sum_{\mathrm{m}\in\mathrm{M}} \alpha_\mathrm{m} \mathbf{p}_\mathrm{m}\right)^\mathrm{T} \left(\sum_{\mathrm{n}\in\mathrm{N}} \beta_\mathrm{n} \mathbf{q}_\mathrm{n}\right)$$

μ is the base score of the predictions. *G,M,N* are sets of global, user, item features respectively, and γ, α, β are global feature, user feature and item feature respectively. Different from general matrix factorization, we want to get the preference order of a user on a list of items. As a consequence, we use pair–wise ranking optimization as optimization criterion

$$\min \sum_{<\mu,k,h>\in D} \ln\left(1 + \mathrm{e}^{-\left(\widehat{y_{\mu,\mathrm{k}}}\right) - \widehat{y_{\mu,\mathrm{h}}}}\right) + \text{regularization}$$

$$\text{Where } \mathrm{D} = \{<\mu, k, h> | k \in Pos(\mu), \mathrm{h} \notin \mathrm{Pos}(\mu)\}$$

$Pos(u)$: the user u gives positive reviews.

In the case, we use training data attributes as items, and only one user who order the +1/-1 rating. It can reach 76.36 AUC with no reduced data and 0.7644 AUC with reduced data in public score board.

I. Gradient Boosting

Gradient boosting [17] is a machine learning technique that assembles some of weak learners. It produces the additive model of the following form: $F(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$, where $h_m(x)$ is so-called weak learner. For each iteration m, gradient boosting tries to find the $h_m(x)$ and γ_m that minimize the sum of the loss function: $\sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) - h_m(x_i))$ and get $F_m(x_i) = F_{m-1}(x_i) + h_m(x)$. The package we take advantage of is GradientBoostingRegressor in scikit-learn [5] . The weak learner that the regressor used is decision tree. We tried the parameters $\mathrm{n_{estimators}} \in [15{,}50]$ and $max_{depth} \in [2{,}5]$ with step size 1. The best choice of parameter is $\mathrm{n_{estimators}} = 32$ and $max_{depth} = 3$ with 0.7669 AUC on the scoreboard.

VII. COMPARISON

A. Naïve Bayes

The naive bayes algorithm is simpler than any other algorithms we mentioned before. We modeled each feature as normal distribution, but they might not be the case. It is worth noting it can handle missing value naturally. However, it didn't perform very well in this experiment, thus it could only serve as a baseline algorithm. As for efficiency and scale, because we assume that the features are independent and follow normal distribution, we only calculate the mean and variance. As a result, the algorithm show short computation time and is able to handle large data.

B. K Nearest Neighbor

k nearest neighbor is another baseline algorithm people tend to use. The training time is very small. But predicting time is long if the data set is large.

C. Linear Model

In this project, we take AUC as measure so only the rank matters. For all linear model we use it is equal to just output $\mathrm{w^T x}$. We try some popular algorithm like linear regression, logistic regression, but in the end we think we could get better in-sample AUC by directly improve AUC by adjusting w. And indeed the latter method doing well. But it do not guarantee to

reach a local maximum but at least it is still worth a try. The training process for Linear regression and Logistic regression is quick compare to other algorithm like svm. The great part about linear model is that VC dimension is low, so we don't need to care about over-fitting so much. For the naive optimize AUC method, the training time can be long, but we may get better in-sample AUC thus better test set AUC.

D. SVM with RBF Kernel

SVM is very popular algorithm. However, in this project, it didn't perform very well. In terms of scale and efficiency, SVM is generally suited for medium size data set. If we use all of the training data to learn a SVM model, it could be extremely time-consuming. As a consequence, we use under-sampling dataset instead. With the help of grid tool and unser-sampling set, we can then perform parameter searching efficiently.

E. Random Forest

Random Forest algorithm is popular because it performs well in recently competitions [6]. In this project, the random forest algorithm is the best individual model that we have performed. Random forest algorithm uses decision tree, and can be parallel computing, so it can scale and train on large data set. The decision tree algorithm is efficient compared to others. In addition, we can compute feature importance to know the importance of each feature in the forest, which can help us dealing with missing values.

VIII. CONCLUSION

Random forest algorithm is the best individual model in our project. Additionally, the algorithm is very efficient. We think the best approach is using EM algorithm to recover the missing values, applying random forest regression and gradient boosting on the under-sampling data, and then blending the two models together. In the case of multiple models blending, we can use multiple models with some different parameters, and try different weighted sum such as normalization, standardization and some linear combination on the scoreboard, we may further improve the result. The best AUC score we achieved in this competition is 0.7711 on the public score board and 0.7546 on the hidden scoreboard.

HOW WE BALANCE OUR WORK

Our work are assigned roughly as below and we always help each other

李文鼎: random forest, gradient boosting, naïve Bayes, Random Optimization with Linear Model

郭佳翰: Support Vector Machine, linear regression, logistic regression, k nearest neighbor

向思蓉: data preprocessing, feature selection, matrix factorization.

REFERENCES

[1] draft CH8 p.280

[2] H.-T. Lin and C.-J. Lin: A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods. Technical report, Department of Computer Science, National Taiwan University, 2003

[3] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[4] Kohavi, R., B. Becker, and D. Sommerfield. 1997. Improving Simple Bayes. In: Proceedings of the European Conference on Machine Learning.

[5] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[6] http://datascience101.wordpress.com/2012/05/31/increase-your-kaggle-score-with-a-random-forest/

[7] J. F. Hair, Jr., B. Black, B. Babin, R. E. Anderson, and R. L. Tatham, Multivariate Data Analysis, 6th ed., Prentice Hall, 2005 Ch.5

[8] Damien Garcia, http://www.mathworks.com/matlabcentral/fileexchange/27994-inpaint-over-missing-data-in-n-d-arrays/content/inpaintn.m

[9] Jolliffe, I. T. (1986). Principal Component Analysis. Springer-Verlag. pp. 487. doi:10.1007/b98835. ISBN 978-0-387-95442-4.

[10] G. Holmes; A. Donkin and I.H. Witten (1994). "Weka: A machine learning workbench". Proc Second Australia and New Zealand Conference on Intelligent Information Systems, Brisbane, Australia.

[11] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall and W. Philip Kegelmeyer, SMOTE: Synthetic Minority Over-sampling Technique, Journal of Artificial Intelligence Research 16 (2002) 321–357.

[12] George H. John and Pat Langley (1995). Estimating Continuous Distributions in Bayesian Classifiers. Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. pp. 338-345. Morgan Kaufmann, San Mateo.

[13] Wikipedia- Random optimization, http://en.wikipedia.org/wiki/Random_optimization

[14] Scikit-learn: sklearn.neighbors.NearestNeighbors http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors

[15] Scikit-learn: Decision Trees http://scikit-learn.org/stable/modules/tree.html

[16] Scikit on GitHub https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/tree/_tree.pyx

[17] Wikipedia – Gradient Boosting http://en.wikipedia.org/wiki/Gradient_boosting

[18] Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen,Zhao Zheng, Yong Yu. SVDFeature: A Toolkit for Feature-based Collaborative Filtering. Journal of Machine Learning Research. 13:3585−3588, 2012